Algorithm Design and Analysis

Homework #3

Due: 1pm, Monday, November 14, 2011

=== Homework submission instructions ===

- Submit the answers for writing problems (including your programming report) through the CEIBA system (electronic copy) or to the TA in R432 (hard copy). Please write down your name and school ID in the header of your documents.).
- Each student may only choose to submit the homework in one way; either all as hard copies or all through CEIBA except the programming assignment. If you submit your homework partially in one way and partially in the other way, you might only get the score of the part submitted as hard copies or the part submitted through CEIBA (the part that the TA chooses).
- If you choose to submit the answers of the writing problems through CEIBA, please combine the answers of all writing problems into only one file in the doc/docx or pdf format, with the file name in the format of "hw3_[student ID].{pdf,docx,doc}" (e.g. "hw3_b99902010.pdf"); otherwise, you might only get the score of one of the files (the one that the TA chooses).
- For each problem, please list your references (they can be the names of the classmates you discussed the problem with, the URL of the information you found on the Internet, or the names of the books you read). The TA can deduct up to 100% of the score assigned to the problems where you don't list your references.

Problem 1. (10%)

We have introduced the concept of paper prototyping in the class; to obtain the tester's opinion, you can create a prototype (of the user interface) using a pen and a piece of paper and discuss with your testers about how your program should work. In this problem, you are asked to implement a paper prototype. Choose a software program or a device that you have used or seen, and come up with an idea to improve its usability. Create a

paper prototype for this improved version of program. Find 3 testers to "test out" your prototype, and gather their opinions/suggestions. At least one of the testers has to be a non-Computer-Science-major person (please specify who they are).
For this problem, you have to submit the following:

- Your paper prototype. It has to be sufficiently detailed (use arrows and short descriptions).
- 3 tester's opinions/suggestions. Please mark the persons who are not Computer-Science-major persons.

Sol: Skipped.

Problem 2. (10%) Solve Problem 15.5-1 on p.403 of the textbook.

Sol:

CONSTRUCT-OPTIMAL-BST($root$)

1 $r = root[1, n]$
2 print "k"$_r$ "is the root"
3 CONSTRUCT-OPT-SUBTREE($1, r-1, r,$"left",$root$)
4 CONSTRUCT-OPT-SUBTREE($r+1, n, r,$"right",$root$)

CONSTRUCT-OPT-SUBTREE($i, j, r, dir, root$)

1 if $i \leq j$
2 $t = root[i, j]$
3 print "k"$_t$ "is the" dir "child of k"$_r$
4 CONSTRUCT-OPT-SUBTREE($i, t-1, t,$"left",$root$)
5 CONSTRUCT-OPT-SUBTREE($i+1, j, t,$"right",$root$)
6 else
7 if $dir ==$"left"
8 print "d"$_{r-1}$ "is the" dir "child of k"$_r$
9 else
10 print "d"$_r$ "is the" dir "child of k"$_r$

Problem 3. (15%) Describe a greedy algorithm that, given a set $\{x_1, x_2, ..., x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Prove that your greedy choice is correct and that the problem has optimal substructure. (Problem 16.2-5 on p.428 of the textbook)

Sol: First we sort the set of n points $\{x_1, x_2, ..., x_n\}$ to get the set $Y = \{y_1, y_2, ..., y_n\}$ such that $y_1 \leq y_2 \leq ... \leq y_n$. Next, we do a linear scan on $\{y_1, y_2, ..., y_n\}$ started from y_1. Everytime while encountering y_i, for some $i \in \{1, ..., n\}$, we put the closed interval $[y_i, y_i + 1]$ in our optimal solution set S, and remove all the points in Y covered by $[y_i, y_i + 1]$. Repeat the above procedure, finally output S while Y becomes empty. We next show that S is an optimal solution.

We claim that there is an optimal solution which contains the unit-length interval $[y_1, y_1 + 1]$. Suppose that there exists an optimal solution S^* such that y_1 is covered by $[x', x' + 1] \in S^*$ where $x' < 1$. Since y_1 is the leftmost element of the given set, there is no other point lying in $[x', y_1)$. Therefore, if we replace $[x', x' + 1]$ in S^* by $[y_1, y_1 + 1]$, we will get another optimal solution. This proves the claim and thus explains the greedy choice property. Therefore, by solving the remaining subproblem after removing all the points lying in $[y_1, y_1 + 1]$, that is, to find an optimal set of intervals, denoted as S', which cover the points to the right of $y1 + 1$, we will get an optimal solution to the original problem by taking union of $[y_1, y_1 + 1]$ and S'.

The running time of our algorithm is $O(n \log n + n) = O(n \log n)$, where $O(n \log n)$ is the time for soring and $O(n)$ is the time for the linear scan.

Problem 4. (10%) The square of a directed graph $G = (V, E)$ is the directed graph $G^2 = (V, E^2)$ such that $\langle u, v \rangle \in E^2$ *iff* G contains a path of length equal or less than two between u and v. Both G and G^2 do not have self-edges. Describe an efficient algorithm for computing G^2 from G when using the adjacency list representation of G. Analyze the running time of your algorithm.

Sol: For each node v in V, start BFS procedure from v until all nodes of distance equal or less than 2 from v have been traversed. We can construct G^2 by adding edges from v to all those traversed vertices into E^2, for each $v \in V$. Since for each $v \in V$, it takes $O(|V|+|E|)$ time to run a BFS procedure rooted at v, the running time is $O(|V|^2 + |V||E|)$.

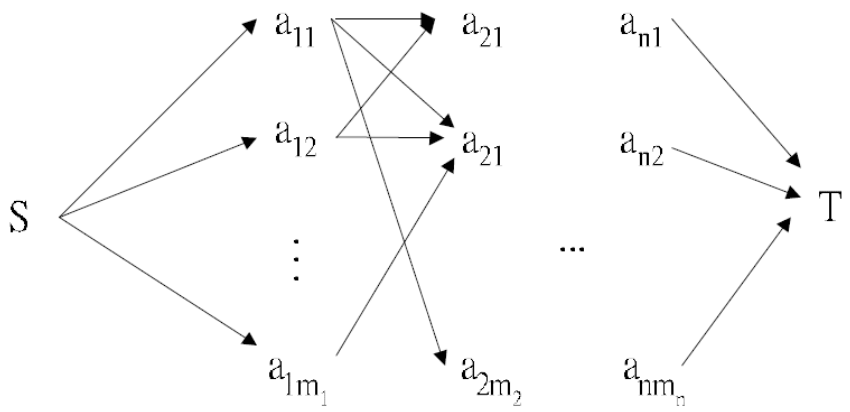

Figure 1: A multigraph

Problem 5. (15%) Consider the multistage graph $G = (V, E)$ as shown in Figure 1. Each edge in G is assigned with a nonnegative weight.

1. (12%) Find a shortest and a longest path between S and T using dynamic programming.

2. (3%) Does your algorithm work if the weights are allowed to be negative? Why?

Sol:

1. It can be proved by cut-and-paste that if $S \to a_{1r_1} \to a_{2r_2} \to ... \to a_{nr_n} \to T$ is the shortest (longest) path between S and T, then for any $1 \leq i < j \leq n$, the subpath $a_{ir_i} \to a_{(i+1)r_{i+1}} \to ... \to a_{jr_j}$ is the shortest (longest) path between a_{ir_i} and a_{jr_j}.

   For example, let $l[1, j]$ denote the minimum length from $a_1 j$ to T and w_j be the length of the path $S \to a_j$, for $j = 1$ to n_1. If the shortest path from S to T passes through a_{1i}, then it must include the shortest paths from a_{1i} to T. Therefore, the minimum length from S to T is $w_i + l[1, j]$.

   Based on this optimal substructure, designing a dynamic programming algorithm updating the length of the shortest path backwardly from T, stage n, stage $n-1$, ..., stage 1 and down to S, we can get the minimum length of the shortest path from S to T in the multistage graph. The procedure of finding the longest path in

the multistage graph is also the same, except that we only keep the value of the maximum length while updating.

2. We allow the weights to be negative in our algorithm, since the principle of the optimal structure is preserved in a multistage graph with negative edges on it.

Problem 6. The outcome of the software company game will contribute to 40% of your homework 3 score and 40% of either homework 4, 5, or 6.

Sol: Skipped.